Robert Kaszubski

CSC 555 Assignment 1

**PART 1**

a) Python File with code attached

211­ = 2048

(24)4 = 65536

44 = 256

85 = 32768

842 MOD 100 = 42

837 MOD 20 = 17

22 MOD 111 = 22

111 MOD 22 = 1

b) Python File with code attached (answers screenshotted below)

V2 – V1 =



V1 + V1 =



|V1| (Euclidean vector length, not the number of dimensions) =



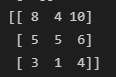
|V2| =



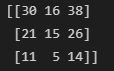
M \* V2 (matrix times vector, transpose it as necessary) =



M \* M (or M2) =



M3 =



c)

HTHT

P(H) x P(T) x P(H) x P(T)

.4 x .6 x .4 x .6 = **.0576**

THTH – should be the same

P(T) x P(H) x P(T) x P(H)

.6 x .4 x .6 x .4 = **.0576**

Exactly 2 heads out of 3

HHT HTH THH are the possible outcomes

3 x (.4 x .4 x .6) = .**288**

Exactly 1 head out of 3

HTT THT TTH are the possible outcomes

3 x (.4 x .6 x .6) = **.432**

d)

i. SELECT PID, Name FROM Project WHERE PID NOT IN (SELECT PID FROM Assign)

ii. SELECT Date, Count(\*) AS Total FROM Assign GROUP BY Date

iii. SELECT PID, Name FROM Project WHERE PID NOT IN (SELECT PID FROM (SELECT PID, Count(\*) as Total FROM Assign) WHERE Total > 2)

Queries for Projects with more than 2 employees assigned to them in the Assign table then returns the projects from Project not in that group based on PID therefore including the projects with 0, 1 or 2 employees assigned to them.

e)

Using a hash function of x mod 15 means that there will be 15 buckets ranging from 0 to 14 as doing mod 15 on any number will fall into that range. 1 to 15 mod 15 will return 0 to 14. So c can be 1 as all are multiples of 1. With the way modulus works, the next number to return this pattern must be 1 + 15. So, then multiples of 16 should also evenly distribute into the buckets. 16 mod 15 will be 1 just like 1 mod 15 is 1. We can keep incrementing this number by 15 to find all possible values of c. So c can be 1, 16, 31, 46, 61, etc.

f)

i. With a replication factor of 3 there are 3 copies of each block. That means if anything happens to 1 block, there is going to be a copy nearby on the same rack. If an entire rack where to fail or if 2 copies failed, then there would be another copy still present with the data on it. If a machine with a block on it were to fail, then a new copy would be made on another machine.

ii. NameNode is responsible for managing the nodes. If a machine were to fail then it would determine which blocks where on that machine, locate the other machines with those blocks, and create a new copy of the block on another machine so there would always be typically 3 (whatever the replication factor is) copies. It regularly monitors replication and handles failures

iii.

The storage cost would be X/128 \* 3 (replications). Except it’s important to note that we would round up to the nearest block in X/128. So, for instance if X is 300mb. Then it divides to about 2.34 or blocks equaling 128, 128, 44. Even though the last block is only 44mb we still allocate the full block to it, so we use 3 blocks in this case times 3 replications so all together 9 blocks of storage.

**PART 2 (Python Code File attached)**

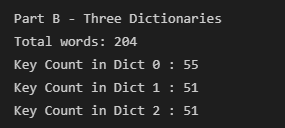
a)

Dictionary has 130 keys



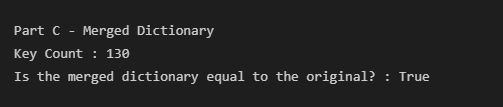
b)

Dictionaries have 55, 51, 51 keys in this run. This changes each time the code is run due to the randomness



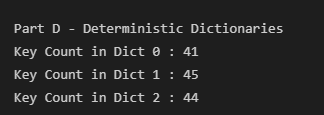
c)

When merged the Dictionary once again has 130 keys and is the same as the original dictionary from part a



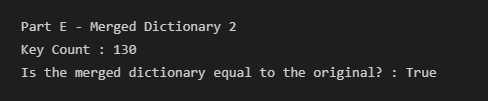
d)

The Dictionaries have 41, 45, 44 keys. Unlike in part b, the number of keys is always the same because we have a method of choosing the dictionary for each word using hash(word) MOD 3



e)

Merging the dictionaries once again results in 130 keys and is the same as the original dictionary



**PART 3**

I used the cdm depaul home page for all of my reading and writing. I downloaded the cdm html file to read from disk several times over the course of about 4 seconds. Then used sys.getsizeof to see how much data was read. This was divided by 10242 to convert to megabytes than divided by the exact time in seconds.

I repeated this using urllib to read directly from the cdm website to determine the read speed over the web. Then I similarly wrote to an empty file for a duration of about 4 seconds to determine the write speed.

a)



b)



c)



d)

Printing out while reading slowed down the speed a little:



Full Console Output with printing:

